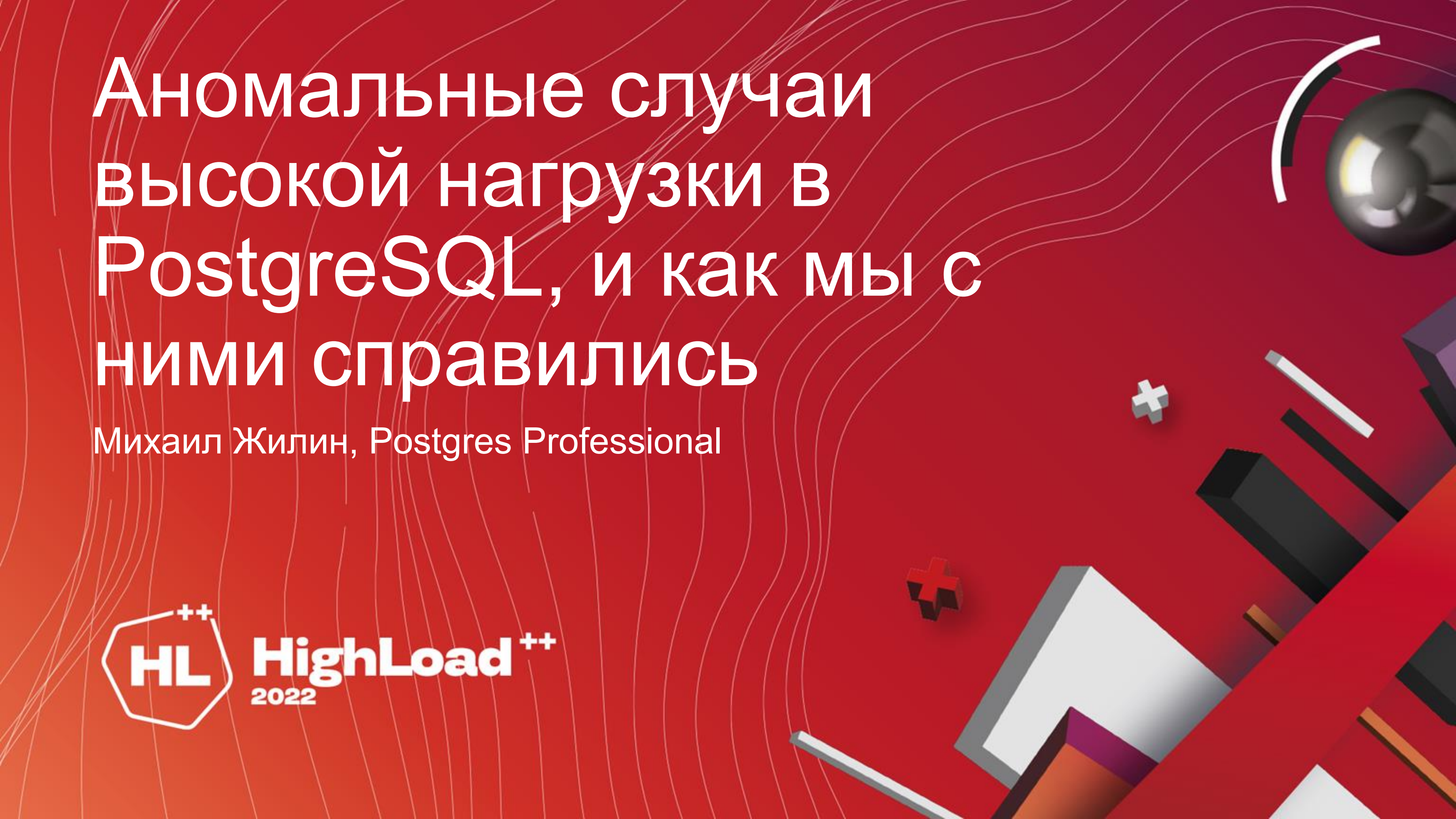

Аномальные случаи высокой нагрузки в PostgreSQL, и как мы с ними справились
Михаил Жилин, Postgres Professional
HL
HighLoad++
2022

Немного о себе



Михаил Жилин

- Инженер производительности
- Контрибьютор в Open Source-проекты

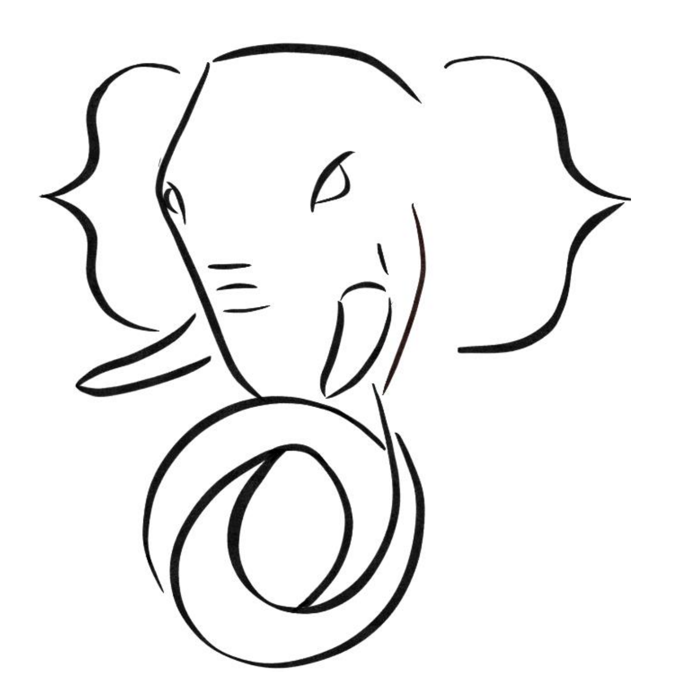
Telegram: @mizhka
e-mail: mizhka@gmail.com





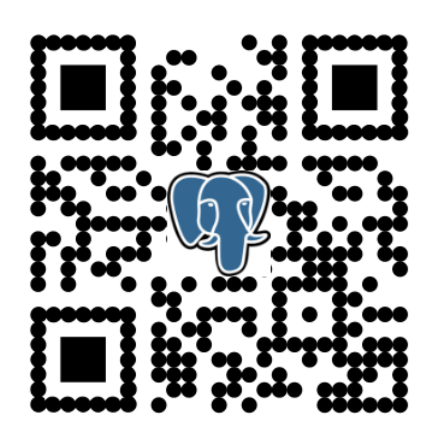
О чём разговор?

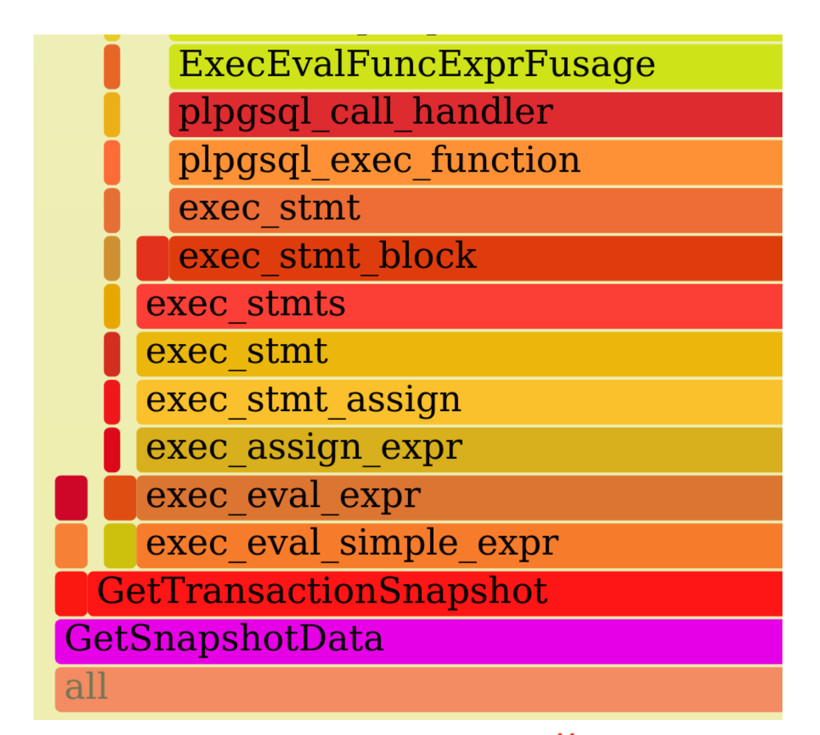
- О PostgreSQL
- О высоких нагрузках из реальной жизни
- Об инструментах профилирования
- О дефектах исправленных
- И неисправленных…



[0] Профилирование

- Perf
- FlameGraph
- Dmitrii Dolgov@PGConf.EU’22 https://bit.ly/3Gp1obg









[0] Профилирование

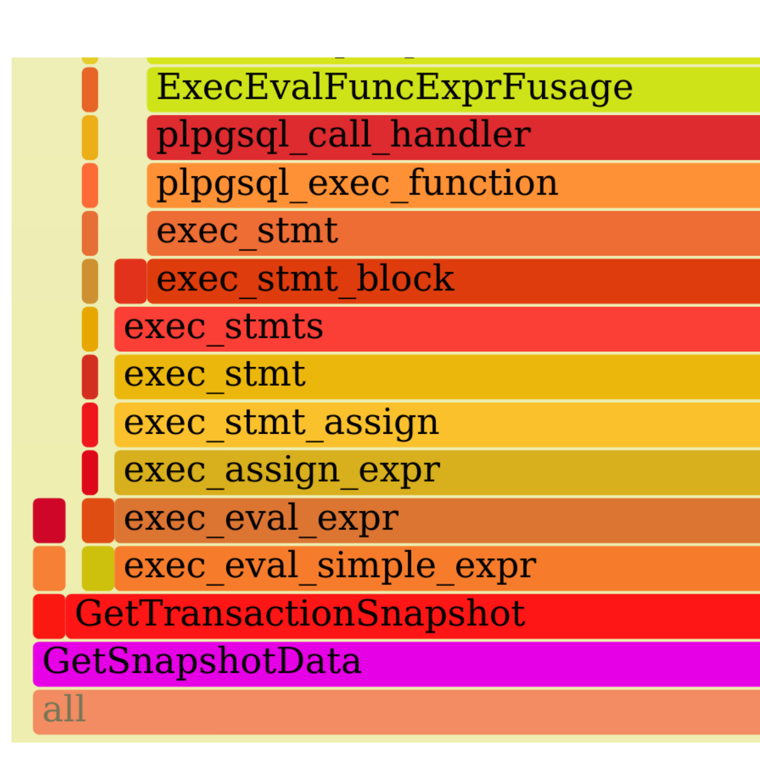
- Быстрый вариант
  - perf top
- Шпаргалка
  - perf record -F 99 -a -g --call-graph=dwarf sleep 2
  - perf script --header --fields comm,pid,tid,time,event,ip,sym,dso
- Требует установки debuginfo для PostgreSQL
- Требует perf-сборку с libunwind (проблема с RHEL-based)





[1] Снимки в MVCC

- **GetSnapshotData**
  - GetTransactionSnapshot
    - **exec_eval_simple_expr**
- plpgsql
- Пример простого выражения
  - i = i + 1
- (<=pg12) Любое plpgsql-выражение требует снимка







[1] Снимки в MVCC

- Transaction Isolation
  - https://www.postgresql.org/docs/current/functions-info.html
  - pg_current_snapshot() -> *xmin:xmax:xip_list*
- Частота вызова: ReadCommitted vs RepeatableRead
- Сложность вычисления
  - Все подключения (до PG 14) или активные транзакции (14+)
  - Требует RW-блокировок





[1] Снимки в MVCC

- Используйте свежие версии PostgreSQL (14+)
  - (<=pg12) Любое plpgsql-выражение требует GetSnapshotData
  - (<=pg13) Тормозит при большом числе idle connections
- Уменьшать число вызовов, к примеру, RepeatableRead



[2] Подтранзакции

- Читающие транзакции начинают тормозить
- Представление pg_stat_activity
    - SubtransControlLock+subtrans или SubtransSLRULock
- savepoint или try-catch
- Подтранзакция = обычная транзакция + parent XID
- Усложняется проверка статуса подтранзакции





[2] Подтранзакции

- ProcArray: 64 подтранзакции на 1 подключение
- pg_subtrans: хранит XID родительской транзакции (до checkpoint’a)
- SLRU: 32 буфера по 8KiB = 65536 подтранзакций
- Домашнее задание: загляните в pg_subtrans, сколько там файлов



[2] Подтранзакции

- Проблемы
  - IO: небольшой размер SLRU
  - CPU: линейный поиск по SLRU
- Такая же проблема у остальных SLRU-кэшей
- Проседание
  - от нескольких %-ов до 10 раз





[2] Подтранзакции

- Workaround
  - постараться не превышать 64 подтранзакций
- https://commitfest.postgresql.org/38/2627/
  - Май 2020: Андрей Бородин
  - Лето 2022: Иван Лазарев и Юрий Соколов
  - slru_buffers_size_scale + оптимизации поиска
  - https://www.postgresql.org/message-id/flat/2BEC2B3F-9B61-4C1D-9FB5-5FAB0F05EF86@yandex-team.ru





[3] Кэш системного каталога

- Долгое планирование
  - pg_stat_statements
  - BIND
- Метаданные + Статистика базы
  - Сохраняется в кэше процесса подключения

```
load average: 71.76, 72.36, 72.54
 0 stopped,    0 zombie
).1 wa,  0.0 hi,  0.9 si,  0.0 st
304 used, 10989072+buff/cache
308 used. 49603376 avail Mem

 %CPU %MEM     TIME+ COMMAND
 70.0  1.1   0:00.38 postgres:(21047) BIND
 65.0  0.5   0:00.14 postgres:(21449) BIND
 60.0  0.9   0:00.19 postgres:(21391) BIND
 60.0  0.5   0:00.12 postgres:(21459) idle
 60.0  0.5   0:00.12 postgres:(21461) idle
 55.0  0.5   0:00.13 postgres:(21421) BIND
 55.0  0.5   0:00.14 postgres:(21431) BIND
```





[3] Кэш системного каталога

- Инвалидация кэша
  - Глобальная и потабличная
  - DDL- и utility-команды
  - On commit рассылает всем процессам **в виде сообщений**

index_getnext_slot
systable_getnext
SearchCatCacheMiss
SearchCatCache3
get_attavgwidth
set_rel_width
set_rel_size



[3] Кэш системного каталога

- Сообщения
  - Циклический буфер из 4096 элементов
  - Обработка при приёме запроса, открытие релейшена и т.п.
  - Кто не успел прочитать, тот идёт в reset



HL HighLoad++ 2022

[3] Кэш системного каталога

- Уменьшать частоту DDL-команд, которые вызывают инвалидации
- Мониторить инвалидации с помощью pgpro_stats: https://postgrespro.ru/docs/postgrespro/13/pgpro-stats#PGPRO-STATS-INVAL-MSGS



[3] Кэш системного каталога

- force_parallel_mode
- debug_discard_caches
- https://postgrespro.ru/docs/postgrespro/15/runtime-config-developer#GUC-FORCE-PARALLEL-MODE

do_page_fault
page_fault
__memset_evex_unali..
[postgres]
hash_create
InitBufferPoolAccess
StartBackgroundWorker





[4] CPU-утилизация в Kernel Space

- **sys >= user**
- Никаких других топовых процессов, кроме postgres

```
top - 18:15:24 up 21 days,  4:04,  3 users,  load average: 79.80, 75.31, 78.49
Tasks: 1418 total,  62 running, 1356 sleeping,   0 stopped,   0 zombie
%Cpu(s): 10.5 us, 87.0 sy,  0.0 ni,  1.3 id,  0.1 wa,  0.5 hi,  0.7 si,  0.0 st
MiB Mem :  64149.7 total,   1638.0 free,  26121.5 used,  36390.2 buff/cache
MiB Swap:   4096.0 total,   1106.3 free,   2989.7 used.  19980.8 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
3009065 postgres  20   0   16608   2508   2236 R   8.8   0.0   0:02.23 netstat
   3661 postgres  20   0  153896   8324   5640 R   7.7   0.0   1015:38 postgres
2972903 postgres  20   0   16.3g 459372 423532 R   7.4   0.7   2:48.01 postgres
2715481 postgres  20   0   16.3g  13.2g  13.2g R   7.2  21.1  78:52.58 postgres
2943744 postgres  20   0   16.3g 478756 443504 R   7.2   0.7   8:07.06 postgres
2961779 postgres  20   0   16.3g 582696 546816 R   7.2   0.9   4:35.96 postgres
2771243 postgres  20   0   16.3g  13.3g  13.2g S   6.9  21.2  46:30.86 postgres
2806845 postgres  20   0   16.3g   2.0g   2.0g S   6.9   3.3  25:20.05 postgres
2816059 postgres  20   0   16.3g  12.4g  12.4g S   6.9  19.8  32:18.11 postgres
2953995 postgres  20   0   16.3g 360120 324068 D   6.9   0.5   3:48.32 postgres
1840331 postgres  20   0   16.3g  15.9g  15.8g R   6.6  25.3  20:30.32 postgres
2976262 postgres  20   0   16.3g  73848  69408 R   6.6   0.1   4:30.96 postgres
2250607 postgres  20   0   16.3g  15.9g  15.8g R   6.3  25.3  16:35.80 postgres
2885227 postgres  20   0   16.3g   1.5g   1.5g D   6.3   2.4  18:22.95 postgres
2891630 postgres  20   0   16.3g 612196 576196 S   6.3   0.9  13:32.72 postgres
```





[4] CPU-утилизация в Kernel Space

- perf top
- [kernel] audit_*
- Отключаем
  - auditctl -e 0

```
Samples: 1000K of event 'cpu-clock:pppH', 4000 Hz, Event count (approx.): 166150960680 lost: 0/0 drop: 0/0
Overhead  Shared Object                          Symbol
  39.78%  [kernel]                               [k] get_mm_exe_file
  21.20%  [kernel]                               [k] fput_many
  12.12%  [kernel]                               [k] audit_exe_compare
   9.78%  [kernel]                               [k] _raw_spin_lock
   0.96%  postgres                               [.] hash_search_with_hash_value
   0.83%  [kernel]                               [k] audit_filter_rules.constprop.21
   0.54%  postgres                               [.] HeapTupleSatisfiesVisibility
   0.53%  [kernel]                               [k] audit_filter_syscall.constprop.17
   0.39%  [kernel]                               [k] copy_user_generic_unrolled
   0.39%  postgres                               [.] heap_page_prune_opt
   0.37%  postgres                               [.] heap_hot_search_buffer
   0.34%  libc-2.28.so                           [.] __memmove_avx_unaligned_erms
   0.30%  [kernel]                               [k] _raw_spin_unlock_irqrestore
   0.24%  postgres                               [.] 0x00000000003515c6
   0.20%  [kernel]                               [k] get_task_exe_file
   0.18%  [kernel]                               [k] unmap_page_range
   0.18%  postgres                               [.] heapgetpage
   0.18%  postgres                               [.] 0x0000000000372da6
   0.16%  [kernel]                               [k] vmxnet3_tq_xmit.isra.63
   0.16%  postgres                               [.] _bt_compare
   0.15%  [kernel]                               [k] finish_task_switch
   0.15%  postgres                               [.] GetSnapshotData
   0.14%  postgres                               [.] 0x0000000000372d83
   0.14%  postgres                               [.] LWLockRelease
```





[4] CPU-утилизация в Kernel Space

<u>30 минут спустя</u>

- **sys** уменьшилось в 8 раз (87->10)
- Избыточные правила auditd

```
top - 18:56:45 up 21 days,  4:45,  4 users,  load average: 11.81, 16.35, 36.34
Tasks: 1684 total,   9 running, 1675 sleeping,   0 stopped,   0 zombie
%Cpu(s): 17.9 us,  9.7 sy,  0.0 ni, 57.7 id, 12.8 wa,  0.6 hi,  1.2 si,  0.0 st
MiB Mem :  64149.7 total,    543.3 free,  25292.7 used,  38313.7 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.  18500.7 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
2751347 postgres  20   0   16.3g  14.9g  14.9g D  26.3  23.8  65:53.20 postgres
2917373 postgres  20   0   16.3g  10.8g  10.8g S  26.3  17.3  17:14.67 postgres
3034048 postgres  20   0   16.3g   3.4g   3.3g R  26.3   5.4   0:38.40 postgres
2816059 postgres  20   0   16.3g  14.6g  14.5g S  26.0  23.3  39:03.23 postgres
2857240 postgres  20   0   16.3g  10.9g  10.8g S  26.0  17.4  28:09.90 postgres
2900147 postgres  20   0   16.3g  10.7g  10.7g S  26.0  17.1  23:13.03 postgres
2943744 postgres  20   0   16.3g  10.7g  10.6g R  26.0  17.0  14:52.92 postgres
3022968 postgres  20   0   16.3g   9.7g   9.6g S  25.7  15.5   2:33.99 postgres
    160 root      20   0       0      0      0 S  17.4   0.0 230:34.78 kswapd0
3039245 root      20   0       0      0      0 I  10.9   0.0   0:29.29 kworker/u3
2941503 postgres  20   0   16.3g   6.2g   6.2g R   8.9  10.0   3:47.02 postgres
2941294 postgres  20   0   16.3g   5.9g   5.9g S   6.2   9.4   3:07.33 postgres
2935380 postgres  20   0   16.3g   6.2g   6.2g S   5.6   9.8   3:23.66 postgres
3003915 postgres  20   0   16.3g   5.5g   5.5g S   5.6   8.7   1:39.58 postgres
                                                        8.9   2:32.95 postgres
```





[5] Highload Observability bug

- **max_connections >= 1000**
- Поток простых запросов (update, select)

```
select wait_event, state, count(1)
  from pg_stat_activity
 where backend_type = 'client backend'
   and state = 'active'
 group by wait_event, state;

 wait_event | state  | count
------------+--------+-------
 ClientRead | active | 5
   <null>   | active | 1
```



HighLoad++ 2022

[5] Highload Observability bug

- WAIT_EVENT_CLIENT_READ
  - Только внутри **secure_read** (часть libpq)
  - “wait until the socket is ready”
  - Ждём нового запроса от клиента

```
▼ secure_read(Port *, void *, size_t) : ssize_t
  ▼ pq_recvbuf() : int
    ▼ pq_getbyte() : int
      ▼ SocketBackend(StringInfo) : int
        ▼ ReadCommand(StringInfo) : int
          ▶ PostgresMain(int, char * *, const char *, const char *) : void
```





[5] Highload Observability bug

- Активный запрос (state = active)
  - Выставляется в начале любого внутреннего действия
  - Снимается перед вызовом secure_read()

- **НЕ МОЖЕТ БЫТЬ АКТИВНЫХ ЗАПРОСОВ С ОЖИДАНИЕМ CLIENTREAD!**





[5] Highload Observability bug

- BackendStatusArray
  - state, query, IP address, timestamps
- ProcArray
  - wait_event
- Информация собирается **неодновременно**

- https://www.postgresql.org/message-id/ab1c0a7d-e789-5ef5-1180-42708ac6fe2d%40postgrespro.ru





[6] Index-Only Scan

- Быстрее, чем Index Scan
- Индексы не хранят xmin/xmax
- Нужно читать их из блока таблицы?
- Есть оптимизация — карта видимости https://postgrespro.ru/docs/postgrespro/15/indexes-index-only-scans



[6] Index-Only Scan

- 2 бита visibility map на 1 блок таблицы
- 1 блок visibility map соответствует 256MiB таблицы
- Index-Only Scan одномоментно умеет работать только с одним блоком visibility map
- Научить работать Index-Only Scan с большим числом блоком VM
  https://www.postgresql.org/message-id/flat/87pn08oast.fsf%40ars-thinkpad





[6] Index-Only Scan

```
before=# explain analyse select * from test order by id;

QUERY PLAN
-----------------------------------------------------
 Index Only Scan using test_idx on test
(cost=0.56..1533125.37 rows=59013120 width=8) (actual
time=0.080..16211.427 rows=59013120 loops=1)
   Heap Fetches: 0
 Planning Time: 0.115 ms
 Execution Time: 18294.049 ms
(4 rows)
```





[6] Index-Only Scan

```
after=# explain analyse select * from test order by id;

QUERY PLAN
--------------------------------------------------------
 Index Only Scan using test_idx on test
(cost=0.56..1533125.37 rows=59013120 width=8) (actual
time=0.056..7522.781 rows=59013120 loops=1)
   Heap Fetches: 0
 Planning Time: 0.076 ms
 Execution Time: 9501.317 ms
(4 rows)
```





[7] Index Scan в NestedLoop’e

- Index Scan во внешней части NestedLoop
- Допустим, есть маленькая таблица и индекс

```
set enable_hashjoin = off;
set enable_mergejoin = off;
set work_mem = 100000000;

select t_small.value
  from t_small, generate_series(1,5000000) i
 where t_small.id = i + 1000
 limit 10;
```



HighLoad++ 2022

[7] Index Scan

```
...
   ->  Nested Loop  (cost=0.14..887512.00 rows=500000
width=33) (actual time=2256.209..2256.210 rows=0 loops=1)
        ...
         ->  Index Scan using t_small_pk on t_small
(cost=0.14..0.16 rows=1 width=37) (actual
time=0.000..0.000 rows=0 loops=5000000)
               Index Cond: (id = (i.i + 1000))
               Buffers: shared hit=5000000

 Planning Time: 0.197 ms
 Execution Time: 2288.133 ms
```





[7] Index Scan

Время 1 запроса (5M index scans)

35000
30000
25000
20000
15000
10000
5000
0

миллисекунды

0 2 4 6 8 10 12

клиенты

latency

PostgresPro



[7] Index Scan

3000
2500
2000
1500
1000
500
0

scans per msec

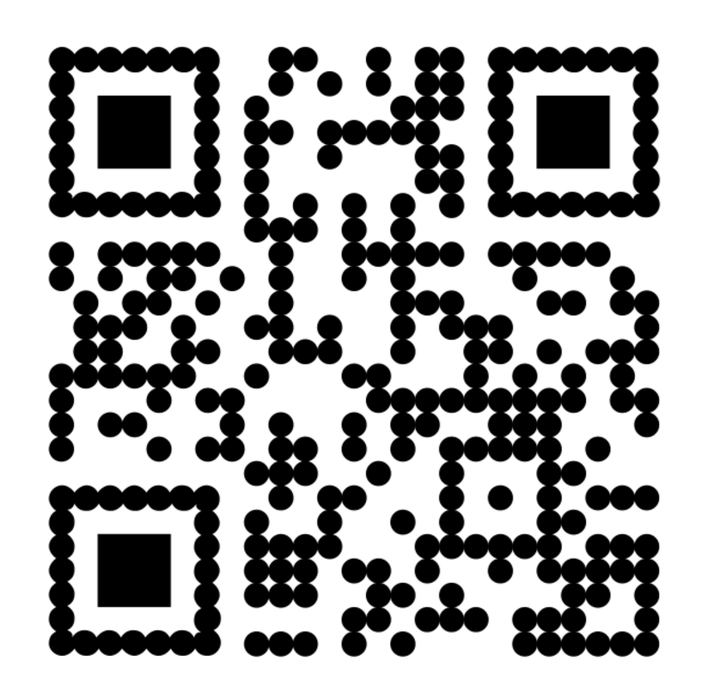
1 2 3 4 5 6 7 8 9 10

clients



В заключение

- PostgreSQL растёт
  - и функционально
  - и по скорости
- Жизнь performance-инженера интересна и познавательна
- All you need is Postgres





Михаил Жилин

Telegram:
@mizhka
E-mail:
m.zhilin@postgrespro.ru



